DOCUMENT RESUME

ED 444 475 IR 020 130

AUTHOR Hamza, Khalid; Alhalabi, Bassem; Marcovitz, David M.
TITLE Remote Labs!
PUB DATE 2000-00-00
NOTE 8p.; In: Society for Information Technology & Teacher Education International Conference: Proceedings of SITE 2000 (11th, San Diego, California, February 8-12, 2000). Volumes 1-3; see IR 020 112.
PUB TYPE Reports - Descriptive (141) -- Speeches/Meeting Papers (150)
EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computer Simulation; Computer Software Development; Computer System Design; *Distance Education; Higher Education; *Internet; *Laboratory Experiments
IDENTIFIERS Java Programming Language; *Learning Environments; Prototypes

ABSTRACT

This paper proposes an Internet facility that will authentically provide laboratory experiments remotely. Such a facility is brought to the doors of the distance learner to help provide learning that is comparable to that offered for conventional students. The paper describes the development of a prototype remote laboratory system, including software, hardware, procedures, and instructional systems. This includes surveys of existing alternatives (mainly software simulation environments), a description of the prototype remote laboratory environment, and a discussion of development issues, such as the reasons behind Java and RMI (Remote Method Invocation) as the system's development tools over other popular alternatives. (Contains 12 references.) (Author/MES)

Reproductions supplied by EDRS are the best that can be made
from the original document.

ED 444 475

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☐ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

Remote Labs!

Khalid Hamza
Center for Innovative Distance Education, Florida Atlantic University
Email: ayah1@concentric.net

Bassem Alhalabi
Center for Innovative Distance Education, Florida Atlantic University, USA
Email: bassem@cse.fau.edu

David M. Marcovitz, Ph.D.
Education Department, Loyola College in Maryland, Baltimore, MD, USA
Email: marco@loyola.edu

G.H. Marks

Abstract: The authors of this paper propose an Internet facility that will authentically provide laboratory experiments remotely. Such a facility is brought to the doors of the distance learner to help provide learning that is comparable to that offered for conventional students. This paper describes the development of a prototype remote laboratory system including software, hardware, procedures, and instructional systems. This includes surveys of existing alternatives (mainly software simulation environments), a description of the prototype remote lab environment developed by the authors, and a discussion of development issues, such as the reasons behind JAVA and RMI as the system's development tools over other popular alternatives.

Introduction

Distance learning (DL) has become a powerful educational tool to reach students at times and places that are not strictly dictated by the educational institution. New technology has made it possible to overcome many of the obstacles of distance in the educational process. Electronic mail, chat rooms, and live interactive video conferencing, for example, help to bring together the people involved in the educational process. The World Wide Web has made available many of the resources that used to be reserved for use on an educational campus (Alhalabi, Hamza, & Sudeep, 1998; Harasim, Hiltz, Teles, & Turoff, 1995; Turoff, 1994).

The success of DL, however, has been problematic for courses that require extensive use of laboratory facilities. In such courses, learning processes (i.e., problem solving and exploration) via laboratory experiments are indispensable. Thus, the distance learner has experienced a void, as no amount of theoretical explanation can be a quality surrogate for real laboratory experimentation. The challenge is to find ways to overcome the obstacle of distance and allow all students to fully participate in laboratory experiments.

In a study that investigated currently available Internet-based educational modalities (Alhalabi, Hamza, & Sudeep, 1998), the authors surveyed a large number of online courses, distance education and virtual universities, and electronic online universities that are involved and actively taking part in DL programs. None of the researched universities and publications discussed or investigated the idea or the concept of real labs via the Internet. Instead, they used primarily simulation software; i.e. virtual labs. Virtual labs, by design, place limitations on students' and teachers' abilities to experiment in real lab environments (Alhalabi, Hamza, & Sudeep, 1998; Hsu, Oge, Hamza, Alhalabi, 1999).

In contrast to the simulated laboratories, this paper provides a conceptual proposal for the implementation of an unrestricted environment where authentic laboratory experiments can take place via the Internet through remote access. Students have unrestricted freedom to apply any inputs and acquire resulting outputs as though they attend the experiment in person. Such a method can liberate students from physically attending the laboratory, and hence, engineering or science experiments can take place through distance learning facilitation software.

IR020130

BEST COPY AVAILABLE

Thus, the authors of this paper, propose an Internet facility that will authentically provide laboratory experiments remotely. Such a facility is brought to the doors of the distance learner to help provide learning that is comparable to that offered for conventional students. This paper describes the development of a prototype remote laboratory system including software, hardware, procedures, and instructional systems. This includes a description of the prototype remote lab environment developed by the authors and a discussion of development issues, such as the reasons behind JAVA and RMI as the system's development tools over other popular alternatives.

Distributed Interface and Protocols

Java Language

Java was chosen for several reasons for the implementation of remote labs. Java allows for programs that can be embedded in Internet web pages. Currently many applications, which are written in a wide variety of languages, including C++, C, and Pascal, have fairly low level facilities like protocol handlers. As remote laboratories rely on the Internet, Java was chosen due to its Internet readiness (Ince & Freeman, 1997).

Java is a machine-independent language, which creates programs that run on a wide variety of computers using a range of operating systems. Since students will log on from a variety of computers to attend the labs, it is more efficient to choose Java, which inherently solves the problem of portability (Flanagan, 1998).

Java programs do not execute directly on the computer and hence will not interfere with the operating system or user data. Instead, they run on the Java virtual machine, solving the problems of security and unauthorized access.

Java is a language well suited for interactive web applications. Most of the applications which are built in other languages, such as CGI and HTML, give the impression of being interactive. But in reality they usually move through a series of text and visual images following pointers to other sections of text and visual images.

Java language offers the feature of multithreading. This means that there are multiple concurrent executions of code, providing a high-level implementation of concurrent processing. This allows many students to work on the same laboratory setup simultaneously (Geary & McCleallan, 1998).

Java is an Object Oriented programming language. It offers facilities to define and manipulate objects, which are self-contained entities that have states and accept messages. Therefore, Java programs can be easily modified, as reusability of code is possible (Cornell & Horstmann, 1997).

Distributed Interface Protocols

Since the virtual online laboratory will be a distributed environment, the implementation had to be in CGI, JDBC, Sockets, IDL, or RMI methodologies. RMI was chosen to build the distributed computing environment, written purely in Java. Other distributed computing environments are compared to RMI below.

The Common Gateway Interface (CGI) is a standard interface that allows web developers to execute programs on an HTTP server. This is a very crude form of client-server computing whereby an HTTP client invokes a program running on the server (usually passing it some data), and the program sends some results back to the client for display on a browser. CGI programs operate in a stateless mode, which means that there is no persistent connection maintained between the client and the server programs. Each time the CGI program gets invoked its runtime environment (memory and system variables) also gets re-established. The program being executed sends the information back to the client and the process shuts down. These recurring starts and shutdowns generate an extra overhead in the server's load and cause delay in the performance. In contrast to CGI, RMI operates in a persistent mode. In other words the network session between the client and the remote objects remains active until the client or the server closes the connection. RMI's garbage collection process automatically closes the TCP stream when the remote object is no longer referenced.

Because of its persistence, RMI is far more effective than CGI in cases where the client is running within the Java virtual machine. The extra overhead is eliminated and the performance is improved.

In some environments, JDBC calls are managed directly by the clients (Reese, 1997). A few JDBC drivers require direct interaction with native database drivers such as the Microsoft ODBC, Oracle SQL Net, etc. The security contained within most commercial browsers prohibits the execution of native code by the Java applets (Cornell & Horstmann, 1997). Other JDBC drivers remove the native calls by introducing an application server. The application server operates as a form of middleware, performing all database interactions and session management on the server. When JDBC calls are incorporated in large-scale applications, such as maintaining student records, it is extremely difficult to maintain the up-to-date database drivers on every client. Also, if either the database engines or the middleware were replaced by another product, every client application that accessed it would require modifications. The JDBC client is responsible for managing the database session and processing the Information. This might cause the network to be burdened with additional traffic as the client repeatedly communicates with the database. In contrast, RMI allows partitioning of data access and workload from the client by encapsulating the JDBC code within a remote object. This approach eliminates added maintenance of the client-side drivers. This also reduces the network utilization by placing the RMI server physically closer to the Database server on the network. In this case a single server was used both as a database and for RMI.

The actual exchange of information between the client and server is performed through the TCP sockets. Architecturally the RMI process operates in a very similar way to those written with Socket objects. Both support persistent network connections between the client and the server (Horton, 1997). In RMI information between the client and server objects is performed through object serialization. However in Sockets one must create the program logic to marshal and unmarshal the objects that are serialized. The RMI is a higher level architecture than sockets. Although the two perform similar functions, the RMI allows you to concentrate on value-added program logic and not network or object transports.

The IDL or Interface Definition Language (Baker, 1997) is a standard defined by the Object Management Group for defining object interfaces. This is intended to establish the object's interface definition regardless of the programming language and environment within which it is implemented. The IDL operates in a model that is similar to the RMI, making use of stubs and skeletons to marshal parameters between objects. RMI is useful in brokering object requests when the objects are written purely in Java.

Although there are many alternatives for building distributed applications like ours, RMI shows its strength in several areas. Since we will be writing the code in Java it is completely portable. It also facilitates application partitioning as the communication takes place across virtual machines. The JDBC package was used as an interface for executing SQL statement (Reese, 1997). We implemented it in this way because: it helps large-scale applications like ours to be written to the JDBC interface without worrying much about which database will be deployed with the application; it keeps the main application insulated from vendor specific issues, so the application can run equally well on Oracle or Microsoft-Access (see Figure 1); the JDBC is built on drivers based on other databases API's, and they are very closely and deliberately mapped to the ODBC counterparts, sharing conceptual components like Driver Manager, Connection, Result Set etc.

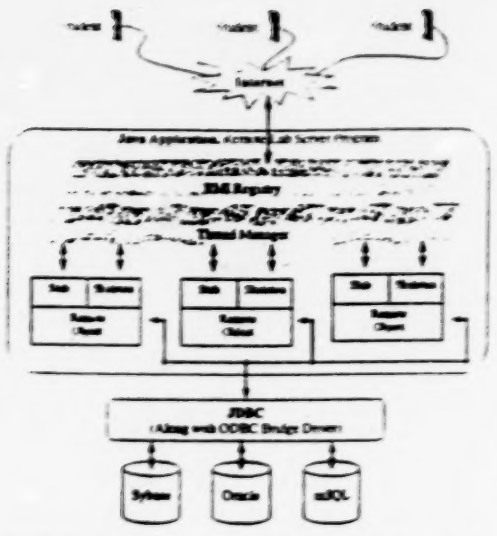

Figure 1: RMI Working Environment Model for the Remote Lab System

System Hardware and Software Model

Hardware Model

For a prototype, we have built a remote laboratory for experimentation with electronic circuits. The 6B Series product family from Analog Devices was used to build the required hardware as it provides a direct interface between the host computer and a variety of analog input, analog output and digital I/O applications. The hardware consists of analog input and output modules and the 6B50 digital I/O board. The following input modules were used in the circuit: the 6B12 Module accepts millivolt inputs from 150mV to 500mV, voltage inputs from 1V to 50V and process current inputs, returning data in engineering units; the 6B21 Module is an analog output module that provides digitally controlled, isolated current loop output to a resolution of 12 bits with an output range from either 0 to 20mA or 4 to 20mA.

System Software Model

The JDBC-ODBC bridge is included in the JDK 1.1 which enables Java applications to access data through drivers written in the ODBC standard. The driver bridge is very useful for accessing data in data sources for which no pure JDBC driver exists (see Figure 1). Through RMI a student can access any other Java object running on a separate virtual machine where the serial communication and the database accesses are performed. The Java applets, on which the students are working, invoke methods contained in remote objects, which issue the calls for serial communication and database access.

The RMI architecture shows the student (client) process and the remote object (a particular experiment) running on separate virtual machines. These two are logically linked to each other through the registry. The Java virtual laboratory application executes as a server process on Virtual Machine A. This application (the remote object), identifies itself with the RMI registry, which is another Java application whose role is to maintain active references to remote objects. A stub is a proxy for the remote object implementing its public interface. When a student object references a remote object, it does so by referencing its stub. A skeleton is

BEST COPY AVAILABLE

the stub's counterpart running on the remote server. Similar to the stub, it provides the mechanism for accepting students' requests, forwarding them to the remote object and sending results back to the students. Through the registry the remote objects offer their services to the students. When the student working on Virtual Machine B wishes to invoke a method contained within the remote object, he/she first communicates with the stub. The stub passes the information to the object reference, which requests the service from the registry. The registry passes the request to the remote object through the skeleton. The student talks to the remote object by accessing the RMI registry on the host where it resides, through a TCP socket connection (i.e., the Internet). Once the registry accepts the student's requests, it delivers a reference in the form of a stub back to the client.

The student now invokes a method contained within the remote object's stub. This invocation may involve exchange of additional objects as parameters to and return values from the function call. Making both the server and the student's applet code serializable makes this communication possible.

At the server end, the parameters, such as the results of the experiment performed and the grade of the student in an experiment, are forwarded to the remote object through a skeleton object. The skeleton dispatches these parameters to the remote object and invokes the appropriate methods. When this function is completed the return value is sent back through the skeleton and the stub objects and is received by the student as shown in Figure 1. The remote object and the student lie at the same level of communication i.e. any request by the student can be answered only by the remote object and vice versa. The same is true for the stub-skeleton and the RMI registry object reference. The Security Manager in RMI (Horton, 1997) also makes sure that the appropriate measures are implemented to make sure that the application is well behaved and does not violate the laboratory rules. With the help of the security manager the students will be prevented from accessing restricted information, such as grades, on the server machine.

Implementation Details

Once the student goes to the URL (http://jupiter.cse.fau.edu/directory.html) the ClientApplet gets loaded (Geary & McCleallan, 1998). The browser now halts and the RMI procedures associated with the browser start by establishing a one to one connection with the Virtual Laboratory server

When the applet runs, it automatically connects to the remote objects because the start method of the applet contains the remote objects' IP address and the remote object registry name. Once the connection has been established, RMI will get the Remote Object Proxy into the users' machine, and this proxy has the reference of the remote object. RMI has server-side skeleton and client-side stub. Once the student invokes any methods the stub interacts with the skeleton and retrieves the required remote object (see Figure 1).

The student can invoke all the methods of the server (remote objects) from the client program with the use of the stub. The server contains the skeleton of the remote objects (stubs and skeletons are generated by the RMIC compiler provided in the JDK kit). The skeleton knows the location of the remote objects in the memory of the server. When the client invokes the method, the stub sends the message to the skeleton; the skeleton calls the remote object; the remote object does the work requested by the client and replies to the server skeleton. The skeleton then replies to the client stub and the stub sends the message back to the client.

Following is an example of student registration into the virtual laboratory. To register, the student types in his/her login name and password and clicks on the register button, invoking the Register() function in the applet, which gaps the login name and password. The OpenDirectory() function will return the thread number for that student (Horton, 1997). Using this thread number the remote object's method is called with the login name and password as parameters. This calls another method for database connection using the JDBC predefined command for SQL query, which checks if any other login name in the database matches the one the user has entered. If it does, a message is sent to the ClientApplet code which asks the user to try another login name. If no duplicate login exists the login name and password are inserted into the database and a positive message will be sent to ClientApplet code which informs the user that registgration has been successful. At the end, PerformLogin() calls the CloseDirectory() function that closes the connection between the client and the remote object (Horton, 1997; Bishop, 1998).

The registered student enters his login name and password and clicks on the Login button invoking PerformLogin() in the applet. The OpenDirectory() function will return the thread number for this particular client. Using this thread number the remote object methods are called and the login name and password are

passed as parameters. This method calls another method for database connection using JDBC predefined command for SQL query. This method checks if the student's login is present in the database. After the check is made, it asks the student to perform Experiment Number 1, if not already performed, or to click on the Get Grade button to see his/her grade in the lab. Finally, PerformLogin() calls the CloseDirectory() function, which closes the connection between the client and the remote object.

If the student has not performed the experiment the student selects Experiment Number 1 from the choice menu and clicks on the Get Experiment button. This will build a new panel, which consists of the new experiment. The Get Experiment button will be enabled only if the student has not performed the experiment. Once the experiment number 1 panel is built the student enters three different values for the current. The student can now click on the Run button, which invokes PerformHardware() in the client applet. This function grabs the current values entered by the student in three different strings. OpenDirectory() will return the thread number for that particular client. The data, which the student has entered needs to be passed to the hardware connected to the serial port of the server. Since the Java Virtual Machine cannot interact with the serial port of the server directly or the operating system, the Win32Port class is developed. The values, which the student enters, are passed as parameters in the function calls to the Win32Port class. The output results are received by the function provided by the Win32Port class methods and these are then passed on to the students as outputs of the experiments. Finally, PerformHardware(), calls CloseDirectory() which will close the connection between the client and the remote object.

After the student has submitted the data to the hardware and gets back the results, the Submit For Grade button is enabled, which will invoke the performSubmission() function in the client applet. This function grabs the values of the currents, entered by the student and the voltage values and calls another method for database connection using JDBC predefined command for SQL query. This will insert all the values along with the student's name in the database. A positive message will be sent to ClientApplet code informing the user that the values have been successfully submitted. Finally, performSubmission() calls CloseDirectory(), which closes the connection between the client and the remote object.

The Hard Part: Java Native Interface (Serial Port Communication)

The last part of completing the package was to write a native interface in Java that talks to the Serial Ports. This consists of Java code (a class) that accesses data through native methods, which are typically particular to a vendor library. Like the bridge driver this class is convenient when a C data access library already exists, but it isn't always very portable.

The Win32Port class implements the access to the computer's communication ports through the Win32 API. The native methods are contained in the JWINPORT DLL, and the Win32Port class is the Java interface to them. The DLL is built using the JNI, which comes with Sun's JDK 1.1.

The standard drivers for serial ports in Windows 95/NT are limited to the ports COM1 through COM4. The compiled DLL adopts this limitation and provides space to handle the four ports. However, the multi-port interface card with a Windows driver makes additional ports visible for the operating system like the standard ports. In this case the DLL has to be recompiled for the number of ports needed.

The port number is passed to each of the methods as parameter rather than having it as a field of the Win32Port class. This is because to access an object field from the C code, a symbolic look up must be made. The port number parameter on the other hand gets passed as a plain integer and works faster.

Conclusion

The authors are successfully progressing through the design and the implementation of the Virtual Lab process over the Internet. Unlike time-shift instruction (experiencing instruction following the live lesson, i.e., videotape, or a software simulation), real labs or real-time instruction (experiencing instruction at some point during the live lesson or experiment) provides students the ability to receive instructions without the teacher's direct presence. Students in Real Lab environments have the freedom to analyze their experiments at a distance. By no means is the student thinking limited but instead his or her complex thinking processes

(basic thinking, creative thinking, critical thinking) are stirred at a distance with out any technical limitation of any kind. A student is still able to conduct an experiment, collect needed information, categorize the information, synthesize the information, and create his or her own conclusions based on data collected, from input and output, without physically attending a lab.

References

Alhalabi, B., Hamza, M. K., & Sudeep, A. (1998). Distance learning and remote real lab experimentation. *Journal of Open Praxis, 2, 24-30.*

Baker, S. (1997). *Corba distributed objects using Orbix.* Reading MA: Addison Wesley Publications.

Bishop, J. (1998) Java Gently. Reading MA: Addsion Wesley Publications.

Cornell, G., Horstmann, C.S. (1997). *Core Java.* Englewood Cliffs, NJ Sunsoft Press Prentice Hall.

Flanagan, D. (1998). *Java in a nutshell.* Cambridge, MA: O'Reilly & Associates.

Geary, D.M. & McCleallan, A.L.. (1998). *Graphics Java.* Englewood Cliffs, NJ Sunsoft Press Prentice Hall.

Harasim, L., Hiltz, S.R., Teles, L. and Turoff, M. (1995). *Learning networks: A field guide to teaching and learning online.* Cambridge: The MIT Press.

Horton, I. (1997). *Beginning Java.* Birmingham, U.K.: Wrox Press, Incorporated.

Hsu, S., Oge, M., Hamza, M. K., & Alhalabi, B. (1999). How to design a virtual classroom: 10 easy steps to follow! *T.H.E. Journal*, 27(2).

Ince, D. & Freeman, A. (1997). *Programming the Internet with Java.* Reading, MA: Addison Wesley Publications.

Reese, G. (1997). *Programming with JDBC.* Cambridge, MA: O'Reilly & Associates.

Turoff, M. (1994). The marketplace road to the information highway. *Boardwatch Magazine* [Online]. Available: http://boardwatch.internet.com/mag/95/apr/bwm47.html